THE BLIND SQUIRREL III

(Above: team photo, from left to right: Jordan Zink, Russell Kittel, Tim Schiefer, Bradley White, Ainsley Baum, Travis Murray, Aaron Huggler, Houston Fortney, Chase Starrett, Fred Donelson (advisor), Robert Ivancic, Ji Hoon Chun)

The Gahanna Lincoln Underwater Robotics Team

Gahanna Lincoln High School, Gahanna, Ohio

"Even a blind squirrel can find a nut sometimes"

Abstract

The Gahanna Lincoln High School (GLHS) Underwater Robotics Team has created a Remotely Operated Vehicle (ROV), affectionately dubbed The Blind Squirrel Three (TBSIII), to compete in the 2010 Marine Advanced Technology Education (MATE) competition. The tasks that TBSIII will complete during this competition include: resurrecting the Hawaii Undersea Geological Observatory (HUGO), collecting samples of a new species of crustacean, taking temperature readings from a hydrothermal vent, and collecting a sample of bacteria.

TBSIII possesses numerous features to accomplish these jobs. Made from PVC, the ROV has the approximate dimensions of 57 cm by 74 cm by 35 cm and masses at 7.9 kg. Two tubes of air and one adjustable nalgene bottle provide floatation for the ROV while foam floats provide the tether with buoyancy. TBSIII employs four lateral and two vertical thrusters for its movement. Furthermore, the team created a PS2 controller that is connected to TBSIII via a microcontroller programmed in C so as to optimize its maneuverability. TBSIII also contains several payload tools to accomplish its mission including: a gripper, hook, temperature sensor, and suction sampler.

The GLHS Underwater Robotics Team spent more than 1000 total hours in designing, building, testing, and modifying TBSIII. Though much of the building process went smoothly, many times the team had disagreements about the ROV's design or had to troubleshoot its systems. TBSIII represents the team's best effort at designing an effective ROV.

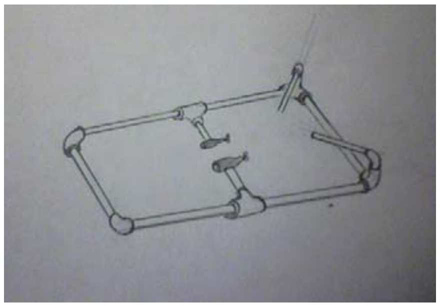
Table of Contents

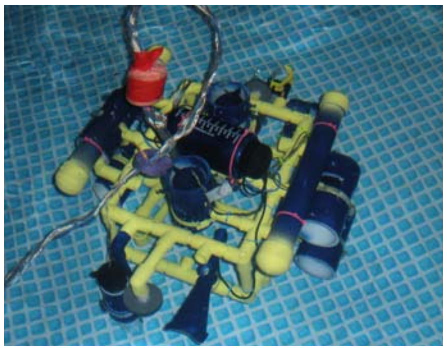
Design Rationale: ROV Components

When the GLHS Underwater Robotics Team created the TBSIII, it decided to plan its design around a set of principles that it believed would create the best possible ROV. These principles included: to condense the ROV’s size, to create a modular design, to build with inexpensive parts, and to create a hydrodynamic, maneuverable robot. Following these guidelines, the team developed several prototypes to test and out of these prototypes, chose one “winner” *(See photo 1).*

Frame

The frame of TBSIII consists of a rectangular prism made of half inch PVC with six ports (unused T-joints) on both its front and back, as well as two pieces of 38 cm, one and a half inch capped off PVC and an adjustable nalgene bottle to provide buoyancy to the ROV *(See photo 2).*


(Photo 1: One of the earliest frame designs)


(Photo 2: The TBSIII)

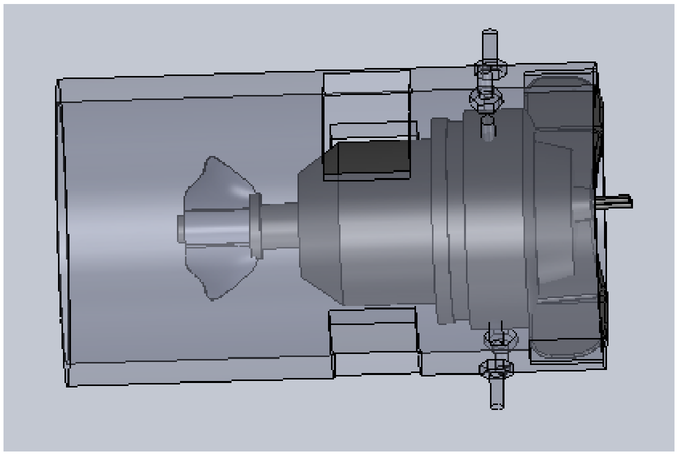
Although from the beginning the club picked PVC as a building material because of its cost, ease of use, and strength, this particular design incorporates several of the most important goals of the GLHS Underwater Robotics Team. First, its condensed, rounded nature makes it hydrodynamic. Also, its lateral thrusters provide a good mixture of torque and speed when turning, increasing its maneuverability. Third, its air-filled PVC buoyancy allows it to travel deeper than previous designs that relied on “noodle” floatation. Most importantly, it provided a total of twelve ports on which to attach tool packages. With this large amount of ports more options existed for where to place and how to design tool packages. In essence, the design fit into the team’s idea of modularity.

Propulsion

Six thrusters propel TBSIII *(See Photos 3 and 4).* Four 1250 Johnson GPH bilge motors control the ROV’s lateral movement while two West Marine Bilge Pro 1000 GPH bilge motors command the machine’s vertical motion. To create the motors, the team attached two blade, two centimeter diameter propellers. Also, to increase the safety of the motors, the GLHS Underwater Robotics Team added motor shields made from cut lengths of three inch PVC pipe around each of the motors.

(Photo 3: A motor in its shielding)

To allow more water to flow past the motors, large rectangular holes were cut into the shielding. With the shields, the 1250 GPH bilge motors spike at 3.5 amps and provide up to 7 N of forward and up to 4 N of backward thrust while the 1000 GPH bilge motors spike at 3 amps and provide up to 5 N of forward and up to 3 N backwards thrust.


(Photo 4: SolidWorks of thruster with transparent casing)

Cameras

TBSIII possesses two X10 Anaconda cameras *(See Photo 5)*. The team chose these cameras for multiple reasons. As far as cost, one can buy X10 Anaconda cameras commercially for 40 dollars each, relatively inexpensive as compared to other underwater cameras. Although the cameras had to be potted by hand in plastic vials with epoxy, the team quickly completed this task and their built-in 60 foot cable made them exceptionally easy to install. Furthermore, the cameras' RCA video jack can easily link them to video cameras to record during missions. Another advantage of the cameras, their ability to image in color, allows objects to stand out better than black and white cameras, making missions easier to complete. To supplement the cameras' ability to view in color, TBSIII possesses two LED lights so that it can perform well in dark areas, such as the conditions one might expect in a cave.

(Photo 5: Potted camera)

The GLHS Underwater Robotics Team placed its cameras so as to maximize their effectiveness. The first and most vital camera is a forward looking and long-range one. The team uses this camera not only to see in front of the ROV but also to see the ROV's gripper and hook. The second camera is placed facing the back of the ROV so that the team can see while the TBSIII drives backwards as well as view the suction sampler and temperature sensor.

Tether

The members of the GLHS Underwater Robotics Team had two goals for TBSIII's tether: to build it inexpensively and to build it out of thin, flexible material for ease of use *(See Photo 6)*.

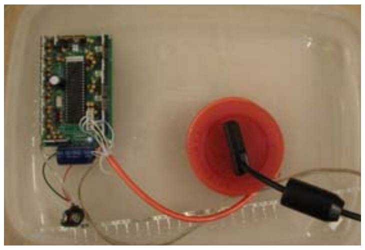
(Photo 6: Team member holding the tether)

In total, it contains eight wires: four 16-gauge speaker wires, two camera cords, a sound sensor wire, and a CAT 5 wire. In all, the tether only possesses a diameter of approximately 2 centimeters. The speaker wires provide power to the ROV's seven motors. The CAT 5 wire allows for several tools to be mounted on the ROV including a temperature sensor and gripper. Foam noodles are used to provide buoyancy. Though teammates considered other methods of floatation, the group decided that these methods would lessen the tether's flexibility or raise its cost, and they subsequently scrapped the ideas.

Controller and Electronics

To achieve better maneuverability, the team chose to use a Playstation controller to drive the robot. The Playstation controller is advantageous because it is ergonomic and provides analog control sticks. Analog control is crucial to the ROV's operation because the motors top speed must be fast enough to travel between locations quickly but have the ability to move slowly for precise control and placement.

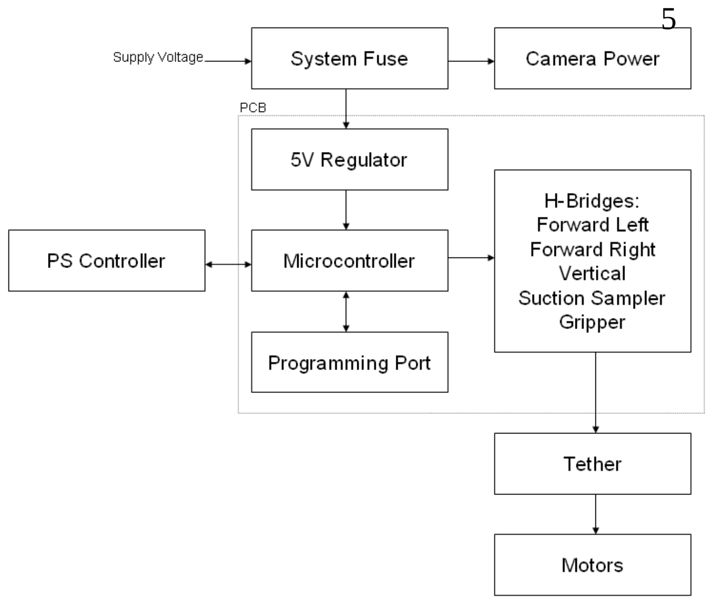
To interface the Playstation controller to the ROV, it was necessary to build a microcontroller to communicate with it and generate variable speeds for the motors. The team decided to use a PIC microcontroller (PIC18F4520 by Microchip) to accomplish this task *(See Photo 7)*. The PIC communicates with the Playstation controller via a serial peripheral interface (SPI) port.



(Photo 7: The PIC)

The PIC controls the motors via five channels of MOSFET H-bridges. Each bridge has four MOSFETs, two N-channel and two P-channel. They operate in pairs. One pair can apply a positive voltage to the motor while the other can apply voltage in the opposite polarity. This pairing enables the PIC to drive the motors both forwards and backwards. To vary the speed of the motor, the PIC generates an oscillating signal. This pulse controls one pair of the MOSFETs to apply either 0 or 12 volts to the motor. The portion of time per cycle that the voltage is turned on (duty cycle) is made short to spin the motor slowly, or long to spin the motor quickly. The frequency of this oscillation is high enough that it is unnoticeable to the user. This method is called pulse-width modulation (PWM).

One decision that the team made was to situate the controller above the water. This means that the controller does not have to be waterproofed, making it simpler and more reliable. But, this method increases the number of unique wires that must be in the tether to control separate channels of motors. If the tether is too large, it will interfere with the operation of the ROV meaning a smaller gauge wire must be used which increases the line's voltage drop because of its increased resistance. Situating the controller on the bottom end of the tether would allow the unique signals to be generated at the end of the tether. This system requires only one, large pair of wires which can be thicker gauge and a few smaller gauge wires to communicate serially with a second control board at the top of the tether. Although this would be a significant improvement, the GLHS Underwater Robotics team decided to rule on the side of reliability for its first year and stay with an above the water system, but because it may want to switch to a below the water system next year, a provision for an RS-422 serial interface has been included as well.



(Diagram 1: Electrical flow chart)

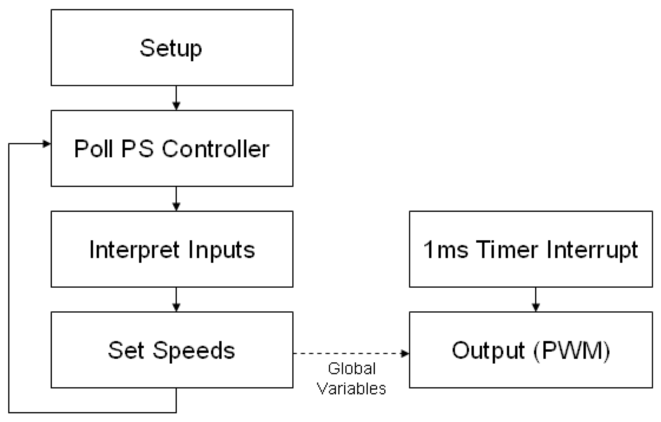
One issue that the team needed to address was the high amount of noise that was generated by the motors. High voltages can damage critical system components such as the PIC and Playstation controller. One way this problem was addressed was through the use of capacitors to limit the rate at which the system voltage can change, reducing the slope and height of voltage spikes. Capacitors were placed on both the battery and regulated voltages. Furthermore, diodes (integrated in the MOSFET packages) provide a short circuit for negative spikes so that they do not interfere with the electronics *(See Schematics 1 and 2 on pg. 7)*.

Software

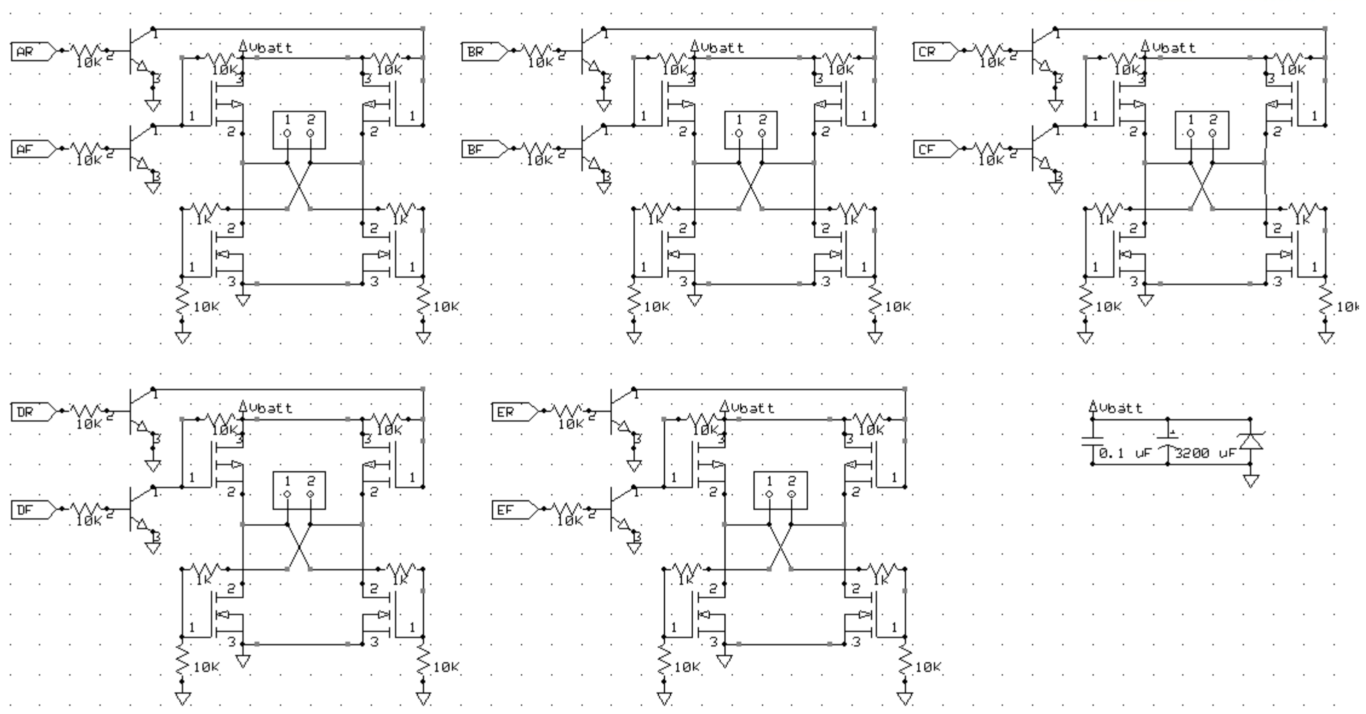
To make it possible to control the ROV with a Playstation controller, it was necessary to understand the interface that is used in a Playstation system. The information about these protocols was found on several web pages *(See References)*. This information was supplemented with data gathered experimentally using an oscilloscope to observe the serial clock and data lines.

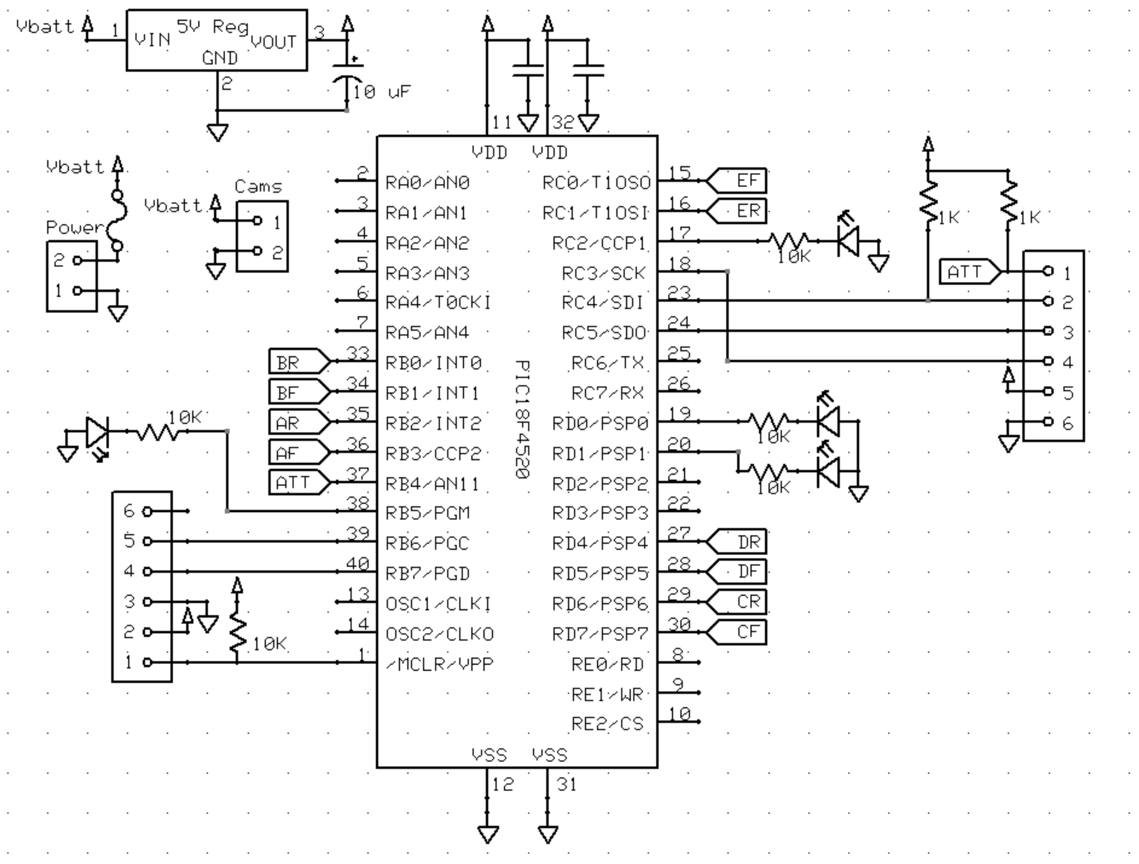
The Playstation controller communicates in eight bit bytes that are sent and received at the same time. The PIC's SPI port was configured for this mode of operation. The PIC is the SPI master that it generates a clock pulse for the communication. The Playstation controller sits idle until data is requested from it. It then sends the data in several bytes that form a packet. Each packet consists of a three byte header and six bytes of information. The first two of these six bytes reports the states of the digital buttons, with a zero indicating that a button is pressed. The latter four of these six bytes reports the positions (x and y) of the two analog joy sticks.

The particular microcontroller that was used did not have enough hardware PWM channels to fit our needs. For this reason, the PWM is generated by the software and outputted on standard digital port pins. In order to maintain a consistent frequency for the pulse generation, a timer-driven interrupt was used. The rest of the program runs in a continuous loop that polls the Playstation controller, determines the appropriate speeds and then, outputs those speeds to the interrupt via a set of global variables.



(Diagram 2: Programming Flow chart)


(Electrical Schematic 1: H-bridges)


(Electrical Schematic 2: Controller)

The software plays an important role in the reduction of noise that is generated by the motors. The largest voltage spikes were observed when the motor was changing directions quickly. To reduce this effect, the software imposes a brief period of no output whenever the user quickly changes direction.

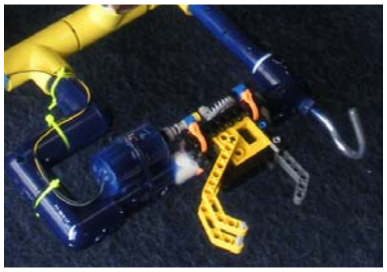
Another important safety function of the software is to ensure that the H-bridges never form a short circuit across the battery. Because the H-bridge has the capability to apply voltage in either direction, it is inherently capable of causing a short circuit. To ensure that this cannot occur, the code is careful about turning off one pair of MOSFETs before turning on the opposite pair. The PIC's clock speed is slow enough that the MOSFETs will have completely switched off before the next line of code. *(See appendix for full code).*

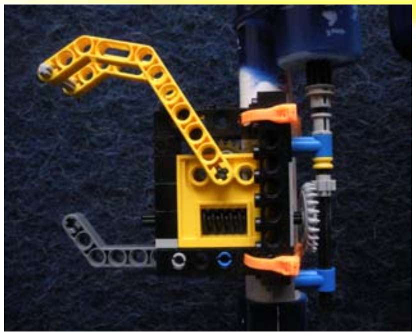
Design Rationale: ROV Tasks

Because of TBSIII's modular design, the GLHS Underwater Robotics Team has no problem quickly hooking on attachments to the ROV's frame to create a robot capable of all the tasks MATE has assigned it. Also, because of its focus on simplicity of design, the team made many of its tools dual-functional.

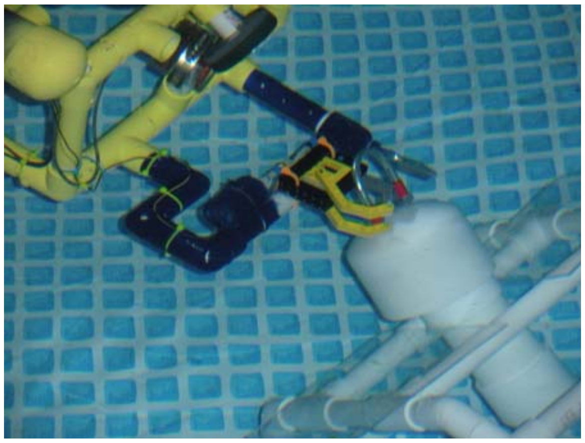
Task 1: Resurrect HUGO

To remove the pins to release the High Rate Hydrophone (HRH) from the elevator, TBSIII possesses both a hook and gripper *(See Photo 8 and 9).*


(Photo 8: Hook and gripper tool package)

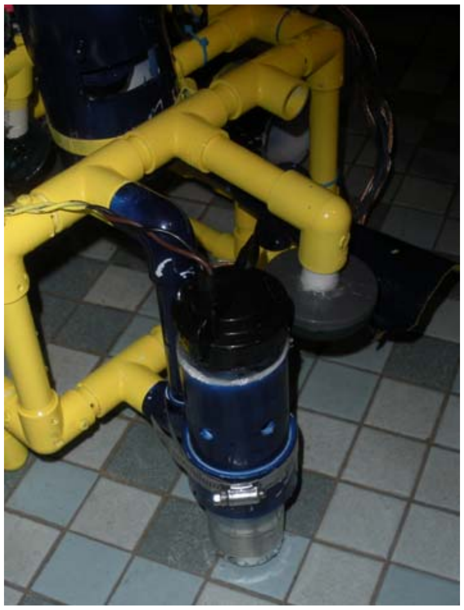

(Photo 9: Vertical view of gripper)

While the gripper, made from LEGOs and a small hobby motor, can pull the pin if it is vertically placed, if the pin has a horizontal orientation, the gripper might have trouble grasping it. The metal hook augments the gripper's effectiveness by allowing TBSIII to unpin the HRH if the pin lies in the horizontal plane. Moreover, the hook acts as a redundant system on the ROV. Even if the gripper short-circuits, the team can still complete its task. After unhooking the HRH, TBSIII will grab it with its gripper and transport it to the site that is rumbling which the ROV will find via its on board sound sensor. The ROV will proceed to HUGO and remove its cap using either its gripper or hook *(See Photo 10).* It will then return to the elevator and retrieve the connector with its gripper and insert it into the HUGO junction box.


(Photo 10: TBSIII removes HUGO's cap with its gripper)

Task 2: Collect samples of a new species of crustacean

This task involves entering an underwater cave to obtain crustacean samples. The suction sampler is the most useful tool package for this task *(See Photo 11)*.


(Photo 11: the Suction Sampler)

The GLHS Underwater Robotics Team created the suction sampler from a 500 GPH West Marine bilge pump that draws water up through the holes on the bottom of an old V8 juice can. Essentially, the suction sampler is an underwater vacuum. Although incredibly simple in design, the team went through several prototypes to achieve this beautiful finished product *(See Photo 12)*. After entering the cave and maneuvering to the back wall, TBSIII will set its suction sampler down upon crustaceans to collect them. It will finish up by maneuvering out of the cave and back to the surface.

(Photo 12: Previous attempts at creating a suction sampler)

Task 3: Sample a new vent site

To complete this task, the TBSIII needs to measure the vent's temperature at three locations, create a graph of height versus temperature, and collect a sample of a vent spire. The GLHS Underwater Robotics Team will employ both its temperature sensor and gripper to achieve these goals. TBSIII's temperature sensor is made from a Vernier temperature probe that has been fitted through both a t-joint and funnel *(See Photo 13)*. Water from the chimney flows into the funnel, over the temperature sensor and up through the t-joint. To collect temperature readings, the ROV simply will approach each chimney and place its temperature sensor as near as possible to the vent and the funnel of the apparatus should direct the chimney's flow onto the sensor *(See Photo 14)*. After the readings are obtained, the team will graph the readings in Excel versus the provided vent heights, 40 cm, 70 cm, and 100 cm.

(Photo 13: Close up of the temperature sensor)

(Photo 14: The temperature sensor in action)

Beyond obtaining temperature readings, the GLHS Underwater Robotics Team will collect a sample spire using its gripper. This gripper, specifically designed to grip many different sizes of PVC, will then transport the pipe to the surface.

Task 4: Sample a bacterial mat

This task requires that TBSIII obtain a sample from a bacterial mat and bring it to the surface and can be achieved with the help of the TBSIII's suction sampler. This suction sampler, specially designed to collect 135 mL of agar and obtain the maximum amount of points, will be placed on top of the bacterial mat. It will then suck up the sample by literally pulling the ROV to the mat's bottom until it hits the bottom of the container. The TBSIII will then rise and return to the surface with its sample.

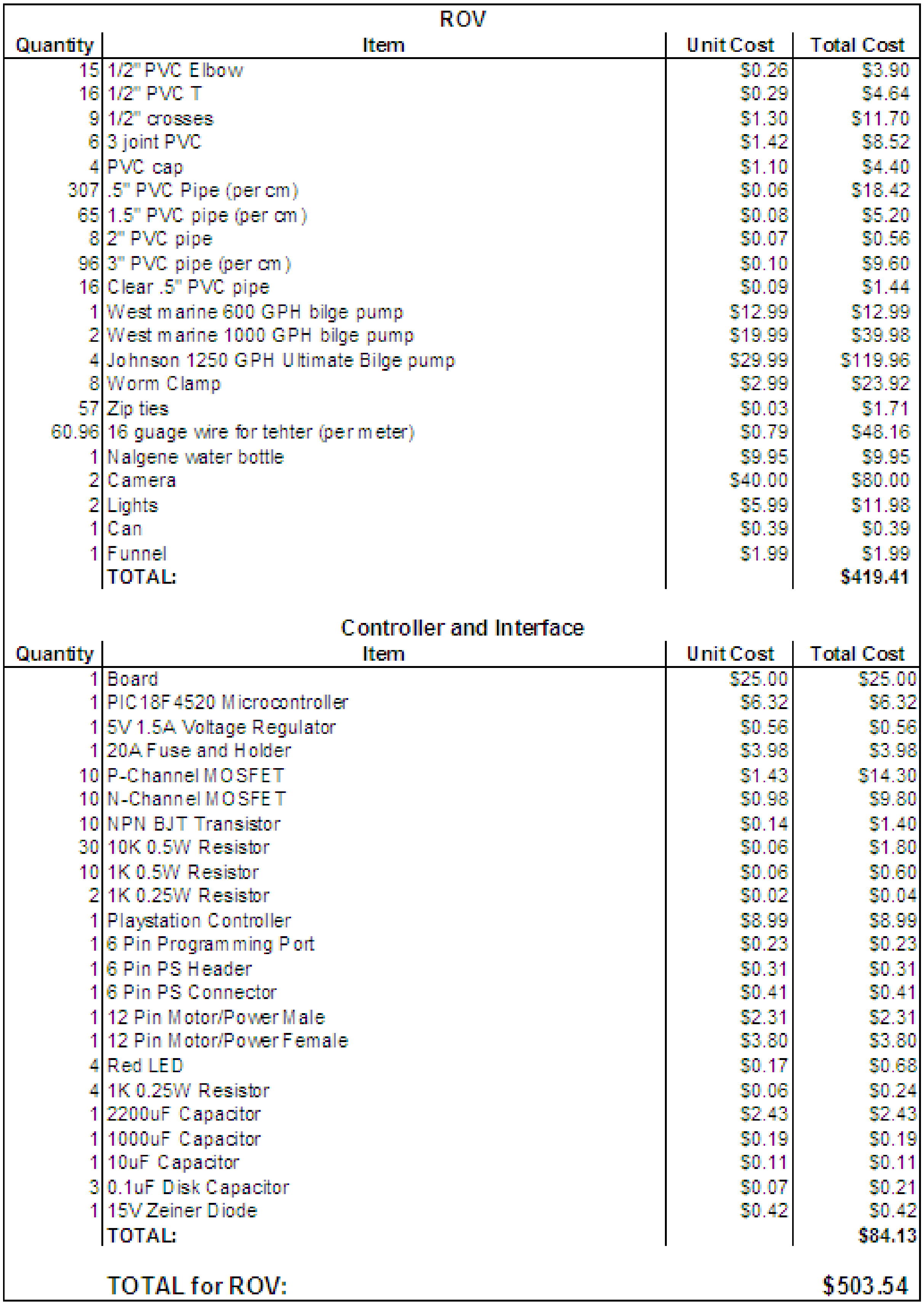
Safety

The safety of TBSIII was a top priority for the GLHS Underwater Robotics Team. To decrease the chance of injury while the ROV is in use, the team's thrusters have been placed in housings. Also, to prevent electric shock, TBSIII's main line contains a 20 amp fuse. Furthermore, because of its complexity, the up-top electronics board contains a fan to cool it. Also, several safety settings exist in the team's computer code so that the board stays cool.

But past the safety features designed into the ROV, the GLHS Underwater Robotics Team has developed the safety checklist that follows to eliminate some of the human error involved in ROV operation.

- Leave battery unclipped while not in use
- Check the bolts on the motors to make sure they are tight before every mission
- Check the propellers to make sure they will not come off before every mission
- Do not touch the ROV while the motors are running
- Drivers should inform the entire team before they start the motors
- Do not touch the electronics with wet hands
- Place a cushion on the suction sampler after every mission to prevent injury when it is out of water

Budget



ROV

Quantity	Item	Unit Cost	Total Cost
15	1/2" PVC Elbow	$0.26	$3.90
16	1/2" PVC T	$0.29	$4.64
9	1/2" crosses	$1.30	$11.70
6	3 joint PVC	$1.42	$8.52
4	PVC cap	$1.10	$4.40
307	.5" PVC Pipe (per cm)	$0.06	$18.42
65	1.5" PVC pipe (per cm)	$0.08	$5.20
8	2" PVC pipe	$0.07	$0.56
96	3" PVC pipe (per cm)	$0.10	$9.60
16	Clear .5" PVC pipe	$0.09	$1.44
1	West marine 600 GPH bilge pump	$12.99	$12.99
2	West marine 1000 GPH bilge pump	$19.99	$39.98
4	Johnson 1250 GPH Ultimate Bilge pump	$29.99	$119.96
8	Worm Clamp	$2.99	$23.92
57	Zip ties	$0.03	$1.71
60.96	16 guage wire for tehter (per meter)	$0.79	$48.16
1	Nalgene water bottle	$9.95	$9.95
2	Camera	$40.00	$80.00
2	Lights	$5.99	$11.98
1	Can	$0.39	$0.39
1	Funnel	$1.99	$1.99
	TOTAL:		**$419.41**

Controller and Interface

Quantity	Item	Unit Cost	Total Cost
1	Board	$25.00	$25.00
1	PIC18F4520 Microcontroller	$6.32	$6.32
1	5V 1.5A Voltage Regulator	$0.56	$0.56
1	20A Fuse and Holder	$3.98	$3.98
10	P-Channel MOSFET	$1.43	$14.30
10	N-Channel MOSFET	$0.98	$9.80
10	NPN BJT Transistor	$0.14	$1.40
30	10K 0.5W Resistor	$0.06	$1.80
10	1K 0.5W Resistor	$0.06	$0.60
2	1K 0.25W Resistor	$0.02	$0.04
1	Playstation Controller	$8.99	$8.99
1	6 Pin Programming Port	$0.23	$0.23
1	6 Pin PS Header	$0.31	$0.31
1	6 Pin PS Connector	$0.41	$0.41
1	12 Pin Motor/Power Male	$2.31	$2.31
1	12 Pin Motor/Power Female	$3.80	$3.80
4	Red LED	$0.17	$0.68
4	1K 0.25W Resistor	$0.06	$0.24
1	2200uF Capacitor	$2.43	$2.43
1	1000uF Capacitor	$0.19	$0.19
1	10uF Capacitor	$0.11	$0.11
3	0.1uF Disk Capacitor	$0.07	$0.21
1	15V Zeiner Diode	$0.42	$0.42
	TOTAL:		**$84.13**

TOTAL for ROV: $503.54

Props

Quantity	Item	Unit Cost	Total Cost
11	1/2" PVC Elbow	$0.26	$2.86
16	1/2" PVC T	$0.29	$4.64
2107	PVC Pipe (per cm)	$0.06	$126.42
2	1/2" crosses	$1.30	$2.60
14	3 joint PVC	$1.42	$19.88
4	45 degree PVC	$0.38	$1.52
1	2" pipe coupler	$0.99	$0.99
12	2" PVC (per cm)	$0.12	$1.44
2	2" PVC cap	$0.80	$1.60
1	1/2" PVC cap	$0.20	$0.20
9	1.5" PVC pipe (per cm)	$0.10	$0.90
1	1.5" PVC cap	$0.67	$0.67
1	Plexiglass	$25.18	$25.18
1	Chicken Wire	$7.99	$7.99
32	zip ties	$0.03	$0.96
	TOTAL:		**$197.85**

Travel/Food/Lodging Costs

Quantity	Item	Unit Cost	Total Cost
11	Plane Tickets to Honolulu (round trip)	$725.55	$7,981.05
6	Plane Tickets to Hilo (round trip)	$146.40	$878.40
5	Plane Tickets to Hilo (round trip)	$160.40	$802.00
4	Hotel Rooms (@111.29/night)	$556.45	$2,225.80
11	Meals on Hilo	$88.00	$968.00
11	Taxi	$20.00	$220.00
11	Travel Day Meals	$20.00	$220.00
	TOTAL:		**$13,295.25**

Income for 2009-2010 School Year

Quantity	Item	Unit Price	Total Price
1	Income from Summer ROV Summer Camp	$425.00	$425.00
1	Donation from IHG Productions	$350.00	$350.00
11	Individual Student payments	$1,088.00	$11,968.00
1	Advisor Donation	$1,253.64	$1,253.64
	TOTAL:		**$13,996.64**

TOTAL COST OF PROJECT:	**$13,996.64**
TOTAL INCOME OF PROJECT:	**$13,996.64**

Troubleshooting Technique

While much of the design process went fairly smoothly, the GLHS Underwater Robotics Team ran into problems when it began to test its new PS2 controller. After the team attached the controller to TBSIII, the ROV seemed to lose a lot of speed. At first, the team believed that the new controller had caused the problem. It took off the new controller and replaced it with a double-pull double-throw switch controller that it had used the previous year. Because this switch did not affect the ROV's speed, the team loosened the bolts that held the motors within the shields to check whether, in an attempt to keep the motors in place, they were lessening the motors' rpm. When this attempt did not work, the team ran a bollard test on the motors with the shields on them and then compared them to a bollard test done before the shields were put on. The tests without the shields had double the force of the tests with the shields. In other words, the team discovered that the motor shields it had placed on the ROV to augment its safety were restricting the motors' flow significantly.

Discovering this problem presented its own conundrum. The motors must have shields on them to keep TBSIII safe but the ROV must also be powerful. So as to compromise, the GLHS Underwater Robotics Team cut large rectangular holes in the shields. These holes allowed more water to flow around the motors and significantly improved the ROV's speed.

Challenges

The GLHS Underwater Robotics Team has faced multiple challenges throughout the building process with both technical and non-technical issues. Because team members are still in high school, they possess many commitments such as sports, clubs, and other after school programs that do not allow them to meet consistently. Furthermore, the unpredictable nature of engineering caused many of the team's initial deadlines to be scrapped. Because of these problems, the GLHS Underwater Robotics Team began to employ the concept of *Flexible Scheduling*. Club activities were held right after school for people not involved in sports and later on for people in them. Some weekend dates were also made available to those who could spend extra time working on the ROV. Although deadlines were still imposed (*See Diagram 3 on pg. 14)*, they were not strictly followed. In fact, each part of the project was scheduled to be finished two weeks before it was actually due, allowing extra time for unexpected problems. *Flexible Scheduling* allowed the team to retain more members and stay organized even if the unexpected did occur.

Yet even more than the interpersonal challenge of keeping a team together, the GLHS Underwater Robotics Team has challenged itself in the technical realm. One of the most important ways that it has done so is by creating a PS2 controller to manipulate its ROV. Although it took many months and several failed attempts, this experience allowed team members to enhance their skills in both electronics and programming. Furthermore, this student-made controller has granted the TBSIII better handling than any of the previous ROVs that the team has built. One of the most difficult problems that the team experienced with the controller was that it would often overheat when one switched from going forwards to going backwards. This problem was alleviated by adding a .3 second delay when switching.

Schedule

Name	September	November	December	January	February	March	April	May	June
Ainsley Baum	Planning, getting to know you, ROV design	Bollard Test	Research	ROV tool packages- brainstorm			Research	Loihi Seamount discusion	Poster and Presentation practice
Ji Hoon Chun		Mathematics of Bollard test						Future Improvement	
Houston Fortney		Electronics		Controller design		ROV toolpackages- gripper	ROV toolpackages- sampler	Software and Electrical Discusion	Driving
Robert Ivancic		Build ROV frame		ROV tool packages- brainstorm	ROV toolpackages- temperature	ROV toolpackages- sampler		Putting final paper together	PresentationP ractice
Aaron Huggler		Build ROV frame	Sheild motors			Perfect the ballast system	"Touch up" ROV		
Travis Murray		Bollard Test	Build ROV frame		ROV toolpackages- hook		Touch up ROV	Driving	Drving
Russell Kittle			Sheild motors						Presentaion Practice
Chase Starrett		Build ROV frame			ROV toolpackages- hook	Perfect the tether	"Touch up" ROV		
Tim Shiefer									
Bradley White							Budget	Budget	
Jordan Zink					SolidWorks, FlowWorks	SolidWorks, FlowWorks	SolidWorks, FlowWorks	SolidWorks, FlowWorks	

(Diagram 3: Schedule)

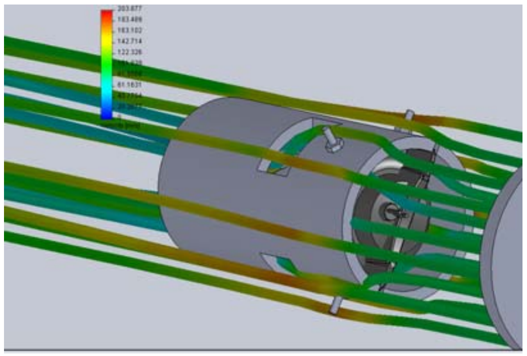
Future Improvements

One improvement that the team could make next year is to include a more pervasive application of modularity in its ROV's design. A fully modular ROV incorporates a small, central base ROV that the team builds and tests early in the year. This type of system would allow the team to work out any basic problems in the ROV before the mission tasks were even announced. Then, when MATE announces mission tasks, the team members can develop tool packages in a relatively short amount of time. Thus, a modular design would speed up the building process and allow more practice days for pilots to gain experience. Also, a modular design would allow for easier testing and modification of individual tool packages because they could more easily be swapped out for other tool packages and therefore, create a more experimental sense in designing the ROV which would allow only the most useful devices to prevail.

Although TBSIII has striven for modularity, it has only been partially successful. The design's main concept as far as modularity was its usage of detachable tool packages. Even though they are "detachable" and can be easily moved from one place to another on the ROV, the tool packages are still attached to TBSIII via their power lines. A system that would allow cords to be easily detached and reattached would allow for a fully modular design and lead to many of the advantages previously mentioned.

Skills Gained

Throughout the process of building the TBSIII, the GLHS Underwater Robotics Team has learned a lot. Through its never-ending quest to improve itself, it has transformed from being mediocre to being exceptional. One way that it has done so is by modeling its ROV in Flow-Works *(See Photo 15)*. Learning to use this software to enhance its designs has improved the team's ability to design hydrodynamic ROVs and therefore has increased the speed and accuracy of the building process. One example of how TBSIII has used FlowWorks is by testing its motors to confirm if the bollard test results for shielded versus unshielded motors were accurate.

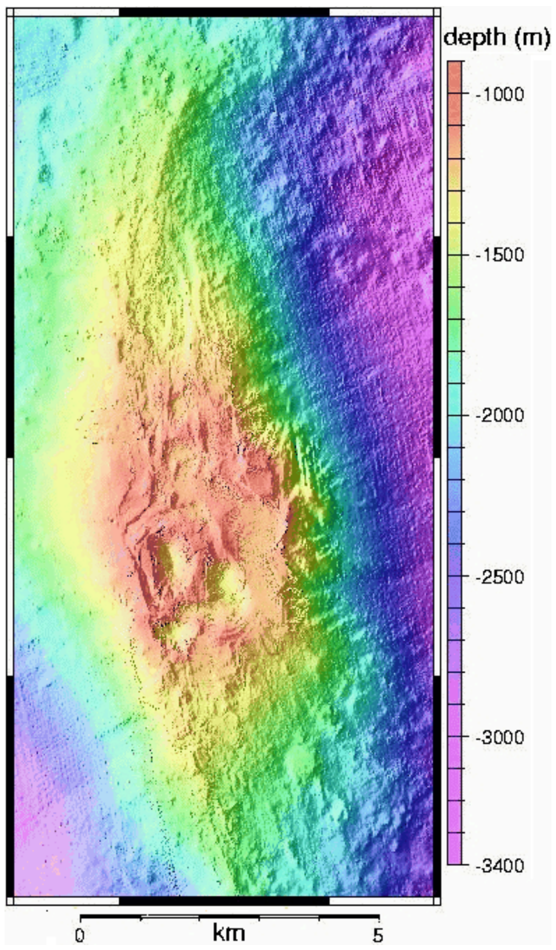


(Photo 15: FlowWorks analysis on thrusters)

But beyond the technical skills the GLHS Underwater Robotics Team gained in building its ROV, it has acquired many interpersonal skills as well. The team learned how to compromise on ideas. If disagreements arose, team members would try both approaches to a problem if possible, and if not, they would vote on which design they liked the most. This approach prevented most bickering about the ROV's design. Unlike previous years, everyone had a voice, and thus, everyone felt as if they could contribute to team decisions.

Loihi Seamount

When first studied, Loihi seemed much like the 80-100 million-year-old Hawaiian volcanoes, old and out-dated. But, upon a 1970 expedition to the seamount, scientists found that unlike previously expected, it was, in fact, rather young and active. Moreover, they discovered that hydrothermal fluids were venting through its summit and the south rift zone, creating layer upon layer of crust until the seamount was formed off of Mauna Loa. The fact that the volcano's surface contained both young and old lava flow remnants was confirmed in 1996 when the first verified eruption of Loihi occurred.



(Photo 16: Topographical map of Loihi, http://www.soest.hawaii.edu/GG/HCV/loihi.html)

In 1997, HUGO was placed on the summit of Loihi. HUGO is useful in detecting the region's geologic activity. But, unfortunately, in early 1998, the HUGO's connector broke. On January 19, 1998, the Pisces V submersible was sent to HUGO to repair a broken connecter that ran to the Junction Box and onto the Big Island of Hawaii, filled with water. After Pisces V fixed the connector, a hydrophone was also added onto HUGO. The hydrophone was placed to listen to the volcano. In February, the hydrophone picked up acoustic sounds coming from the volcano, which caused suspicions of an eruption. The MATE 2010 competition parallels this mission. One of the tasks during the competition that the teams must complete starts with a platform. This platform contains the HRH connected to it, which must be removed from the platform and placed in an area that is suspected of seismic activity. Also, the connector must be placed inside the HRH, which simulates the fixing of the connector on HUGO in 1998. A microphone is located on the ROV so that the ROV can sense seismic sounds under water, much like the hydrophone does on the connector on HUGO.

Reflections

Ainsley Baum: Lead Researcher and Builder

As a first year member of the underwater robotics team, I have learned the meaning of teamwork after spending months of hard-work, time, and effort into our ROV. Never having been on the team, I came into the club not knowing what to expect, but the guys really welcomed me and encouraged me to help out in any way possible even with my lack of experience. The members utilized my ability to help, and I acted as a spare hand to help build the ROV, I ran a bollard test and assisted in the maintenance of the robot. I even got to research more about Loihi. Overall, I feel as if I have contributed a lot to the team.

Ji Hoon Chun: Photographer, Mathematician, and Theoretician

Shortly after the 2009 competition finished, the team decided to make the design of the ROV in a modular format. At that point in time, I knew that it was a concept that I would mesh well with. Over the next year, I focused on this concept in both the theoretical and practical sense. I helped with testing motors fitted with different shields, independently of the rest of the ROV. Like last year, I looked at the Underwater Robotics project with a mathematical eye, and the data graphs I made gave useful info on motor shielding. Ever since I joined the Underwater Robotics team, I took pictures of the team working on the ROV, as well as the ROV itself, and this year was no exception. I've greatly enjoyed my year here.

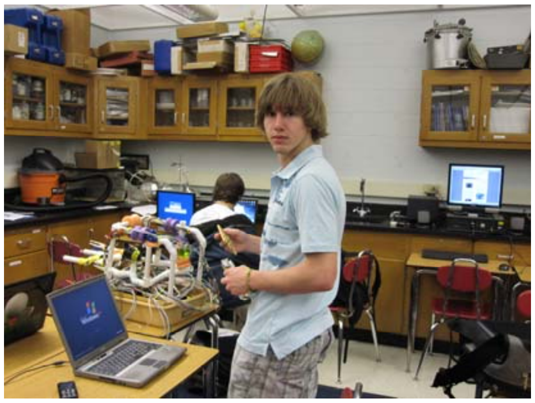
Houston Fortney: Electrician and Programmer

I learned a lot from participating in this experience. I greatly expanded my knowledge and understanding of electronics and software. I accomplished my lifelong goal of laying out and populating my own printed circuit board. I also learned a lot about perseverance and troubleshooting. This was especially true on one particular event when everything stopped working and my entire control board became very hot, burning my fingers and fatally damaging virtually every expensive component in the system. It took a lot of time to determine the cause of these troubles, but I did not want to let my team down, so I worked through the frustration. Furthermore, I learned a lot about developing something until it works as well as you would like it to. This was especially true of our approach to collecting the sample of auger. We went trough many prototypes from straws to soup cans that included check valves or even mouse traps. Finally we created a device which worked quickly and reliably at a very low cost. I've learned how engineering is accomplished in the real world.

Robert Ivancic: Captain and Editor

Being captain of the GLHS Underwater Robotics team has been a unique challenge. Instead of contributing directly to build the ROV, I took on a more overarching role and unlike previous years, had the authority to tell my teammates what to do. As captain, I tried to make everyone on the team feel as if they had a place and could contribute to the project. Overall, I think that the team has improved exponentially this year both in their ability to work together and in their technical knowledge, and I'm excited for the competition.

Aaron Huggler: Chief Builder



My overall experience in working with TBSIII has been really good. My task was to help design and build the ROV and drive it. Because I had worked with most of the members on the team in my previous year of ROV and in other classes, I was already very familiar, and we bonded well.

Travis Murray: Driver

Overall, this year has been an exciting and interesting journey on our way to the competition in Hawaii. I have enjoyed the challenges as well as the exhilarating experiences that have occurred this year during preparation for the competition. Some difficulties that we faced included teamwork on some projects, as well as communication and decision making on rover designs. These teambuilding and team working skills are very useful and will help me in the future for job projects. I am very confident in our team and I am excited to compete in Hawaii.

Russell Kittle: Chief Designer

My role in this project was designing the ROV. I gave people ideas on what to do for the ROV and how to solve problems that occurred during the development project. This project was a great experience for me. I received a lot of creative freedom with the project and leadership. Even though I had track and was not able to be there for some time, I still could take a great interest in robotics. This project gave me a sense of how people come together to accomplish a goal and how amazing that feeling can be.

Chase Starrett: Builder

This year has been really fun and interesting for me. Last year, I wanted to work on the ROV but due to my participation in Lacrosse my schedule never matched up, and I wasn't able to participate fully. This year, I decided to give up lacrosse and focus on my school work and underwater robotics. I'm glad that I did. Designing and building a ROV was a great experience. The team worked really well together, and I'm really enjoyed of the fact that we could just throw out ideas and build upon them from there. Overall, I had a great time building a ROV and working with our team. I feel that we worked really well together, and I hope we will still be able to work well together during the competition to complete all of its challenges.

Tim Schiefer: Builder

Working as part of this underwater robotic team has been a great experience. It has been a rewarding experience working with my teammates. There have been some different opinions in how to best accomplish our assigned tasks. After seeing those ideas become a reality, I can't wait to be in the actual competition. I am confident we will do better this year than we have in the past.

Bradley White: Budgeter and Builder

This year has been a great experience to me. Like last year, I am happy to work with my teammates to create a ROV and to become a part of something larger than myself. I have grown throughout the year by being involved in many different aspects of the ROV as well as watching over the new members as they learn the skills to design a ROV. Overall, I feel as though the MATE competitions have helped me prepare for the real world that lies after graduation.

Jordan Zink: SolidWorks and Builder

Through this entire project, I have acquired a lot of knowledge about the general construction of robots. While the project specifically dealt with an Underwater ROV, many of the techniques we learned can be extrapolated to other robots. Prior to this project, I did not build with materials other than LEGOs and knew very little about electronics. While building the ROV, I learned to create different things with PVC. Also, I learned much about soldering and controllers. Most importantly, I have learned that things don't have to be perfectly precise and correct. Things will still work even if they are not exactly accurate. Breaking my previous mindset of precision has been hard, but this ROV has allowed me to do so. Overall, I feel as if I have widened my horizons by participating in this project.

Acknowledgements

Many people and organizations have helped the GLHS Underwater Robotics team create the TBSIII. The team would like to thank the Gahanna-Jefferson Education Foundation for making a generous financial donations to the project. Furthermore, it would like to thank MATE for providing a venue to gain skills in underwater engineering. The team wants to thank both MATE and SolidWorks for supplying the SolidWorks software.

On a more personal level, the GLHS Underwater Robotics team would like to thank Matt Gardener for answering its questions. It also owes a great thanks to Clark Fortney whose advice was instrumental in designing the ROV’s electronics. Moreover, the entire team wishes to thank its advisor Fred Donelson, who put in countless hours and some of his own money to help lead the team to success and whose moral support has guided the team even through its toughest times.

References

Electronics

Blanchard, Eugene. “A better MOSFET H Bridge Schematic.” The Using Mosfets Website. August 2007. Web. 17 Dec. 2009. < http://www.cadvision.com/blanchas/hexfet/np-s.htm>

“Interfacing with a PS2 Controller.” Curious Inventor. 2008. Web. 21 Dec. 2009. < http://store.curiousinventor.com/guides/PS2/>

McCubbin, Andrew. “Sony Playstation Controller Information.” 13 Aug. 1998. Web. 15 Dec. 2009. < http://www.gamesx.com/controldata/psxcont/psxcont.htm>.

Research

"Loihi Submarine Volcano: A Unique, Natural Extremophile Laboratory." NOAA's Office of Oceanic and Atmospheric Research. 18 Dec. 2000. Web. 19 May 2010. <http://www.oar.noaa.gov/spotlite/archive/spot_loihi.html>.

Rubin, Ken. "Loihi." SOEST | School of Ocean and Earth Science and Technology. 19 Jan. 2006. Web. 18 May 2010. <http://www.soest.hawaii.edu/GG/HCV/loihi.html>.

Appendix: Programming

```c
/*****************************************************************************
Code for Underwater Robot Controller
Gahanna Lincoln High School 2010
Author: Houston Fortney

    This program reads data from a Playstation Controller via the SPI
port.  This information is used to control 5 motors via H-Bridges
driven by inturrupt-based pulse width modulation.

Roles of Functions:
    main: Polls PSController and processes motor speeds.  Outputs desired
    speeds to BBPWM.

    Setup: Sets all relevant registers to desired settings.

    PSConfig: Sends the PSController a series of packets that forces it
    into analog mode.

    InterruptServiceLow: Identifies scource of interupt and calls
    appropriate function.

    BBPWM: Called by Timer0 interrupt.  Gerates desired duty cycle.

    Flip: Corrects the incoming PSController bytes from LSB first to MSB
    first.

    PSController_TX_RX: Sends and recieves one byte from the PSController
    via the SPI port in full duplex.

Motor Channels:
    Motor A: Right Trusters
    Motor B: Left Trusters
    Motor C: Vertical Thrusters
    Motor D: Suction Sampler Pump
    Motor E: Gripper Drive Motor

int Gahanna;
while (1)
{
    Gahanna = 0x01;
}

****************************************************************************/

/** C O N F I G U R A T I O N   B I T S ******************************/
#pragma config OSC = INTIO67, WDT = OFF, PBADEN = OFF, LVP = OFF

/** I N C L U D E S **************************************************/
#include "p18f4520.h"
```

```c
/** G L O B A L   V A R I A B L E S ********************************/
char MotorAOut = 0;                         //Speed of Motor A (-32 to 32)
char MotorBOut = 0;                         //Speed of Motor B (-32 to 32)
char MotorEOut = 0;                         //Speed of Motor E (-32 to 32)

/** L O O K - U P   T A B L E S ****************************************/
//LinearSpeed: From 0:255 to -32:32
static rom char LinearSpeed[] = {
-32, -32, -31, -31, -31, -31, -30, -30, -30, -30, -29, -29, -29, -29, -28, -28,
-28, -28, -27, -27, -27, -27, -26, -26, -26, -26, -25, -25, -25, -25, -24, -24,
-24, -24, -23, -23, -23, -23, -22, -22, -22, -22, -21, -21, -21, -21, -20, -20,
-20, -20, -19, -19, -19, -19, -18, -18, -18, -18, -17, -17, -17, -17, -16, -16,
-16, -16, -15, -15, -15, -15, -14, -14, -14, -14, -13, -13, -13, -13, -12, -12,
-12, -12, -11, -11, -11, -11, -10, -10, -10, -10,  -9,  -9,  -9,  -9,  -8,  -8,
 -8,  -8,  -7,  -7,  -7,  -7,  -6,  -6,  -6,  -6,  -5,  -5,  -5,  -5,  -4,  -4,
 -4,  -4,  -3,  -3,  -3,  -3,  -2,  -2,  -2,  -2,  -1,  -1,  -1,  -1,   0,   0,
  0,   1,   1,   1,   1,   2,   2,   2,   2,   3,   3,   3,   3,   4,   4,   4,
  4,   5,   5,   5,   5,   6,   6,   6,   6,   7,   7,   7,   7,   8,   8,   8,
  8,   9,   9,   9,   9,  10,  10,  10,  10,  11,  11,  11,  11,  12,  12,  12,
 12,  13,  13,  13,  13,  14,  14,  14,  14,  15,  15,  15,  15,  16,  16,  16,
 16,  17,  17,  17,  17,  18,  18,  18,  18,  19,  19,  19,  19,  20,  20,  20,
 20,  21,  21,  21,  21,  22,  22,  22,  22,  23,  23,  23,  23,  24,  24,  24,
 24,  25,  25,  25,  25,  26,  26,  26,  26,  27,  27,  27,  27,  28,  28,  28,
 28,  29,  29,  29,  29,  30,  30,  30,  30,  31,  31,  31,  31,  32,  32,  32};

/** F U N C T I O N   P R O T O T Y P E S ***************************/
void main (void);
void Setup (void);
void PSConfig (void);
void InterruptServiceHigh (void);
void InterruptServiceLow (void);
void BBPWM (void);
unsigned char Flip (unsigned char Old);
unsigned char PSController_TX_RX (unsigned char DataWrite);

/** I N T E R R U P T   V E C T O R S *******************************/
//High priority interrupt vector
#pragma code InterruptVectorHigh = 0x08
void InterruptVectorHigh (void)
{
  _asm
    goto InterruptServiceHigh                   //Jump to interrupt routine
  _endasm
}

//Low priority interrupt vector
#pragma code InterruptVectorLow = 0x18
void InterruptVectorLow (void)
{
  _asm
```

```
//Check for inversion
if ((DigitalOne & 0b00010000) == 0x00)
//Looks for Start
{
    Invert = -Invert;                    //Flip inverter
    LATDbits.LATD1 = !LATDbits.LATD1;    //Flip inversion indicator LED
    for (i = 0; i <= 8000; i++);         //Debounce
}

//Process Motor A input
MotorARead = Flip(MotorARead);           //Fix MSB
MotorA = LinearSpeed[MotorARead];    //Use look-up table

//Process Motor B input
MotorBRead = Flip(MotorBRead);           //Fix MSB
MotorB = LinearSpeed[MotorBRead];    //Use look-up table

//Process Speeds
if (Invert == -1)                        //Check for inversion
{
    MotorASpeed = -MotorB;
    MotorBSpeed = -MotorA;
}
else
{
    MotorASpeed = MotorA;
    MotorBSpeed = MotorB;
}

//Output Motor A (Left)
//Look for sign change
if (((MotorASpeed <= 0) && (MotorASpeedPrev > 0)) ||
    ((MotorASpeed >= 0) && (MotorASpeedPrev < 0)))
{
    MotorAEnable = 0;                    //Delay
}
MotorASpeedPrev = MotorASpeed;           //Update previous speed
if (MotorAEnable >= 100)
{
    MotorAOut = MotorASpeed;             //Use desired speed
}
else
{
    MotorAOut = 0;                       //Turn motor off
}
```

```
    goto InterruptServiceLow                    //Jump to interrupt routine
  _endasm
}

#pragma code
//-----------------------------------------------------------------------
//Main program
void main (void)
{
    //Declarations
    char Invert = 1;                            //-1 if inversion is desired
    unsigned char MotorARead = 0x00;            //Motor A Serial Read
    unsigned char MotorBRead = 0x00;            //Motor B Serial Read
    char MotorA = 0;                            //Processed speed of Motor A
    char MotorB = 0;                            //Processed speed of Motor B
    char MotorASpeed = 0;                       //Motor A intermediate
    char MotorBSpeed = 0;                       //Motor B intermeidate
    char MotorASpeedPrev = 0;                   //Last Speed of Motor A
    char MotorBSpeedPrev = 0;                   //Last Speed of Motor B
    char MotorCPrev = 0;                        //Last Direction of Motor C
    char MotorEPrev = 0;                        //Last Direction of Motor E
    unsigned int MotorAEnable = 100;            //Enabled if greater than 100
    unsigned int MotorBEnable = 100;            //Enabled if greater than 100
    unsigned int MotorCEnable = 100;            //Enabled if greater than 100
    unsigned int MotorEEnable = 100;            //Enabled if greater than 100
    unsigned char DigitalOne = 0x00;            //Digital serial read for byte one
    unsigned char DigitalTwo = 0x00;            //Digital serial read for byte two
    unsigned int i;                             //Looping variable

    Setup();                                    //Configure controller

    for (i = 0; i <= 50; i++);                  //Startup delay (for serial)

    //Main program loop
    while (1)
    {
        //Poll Packet
        LATBbits.LATB4 = 0;                     //Drop ATT line to begin transmission
        PSController_TX_RX (0x80);              //Start packet
        PSController_TX_RX (0x42);              //Header
        PSController_TX_RX (0x00);              //Header
        DigitalOne = PSController_TX_RX (0x00); //Read digital byte 1
        DigitalTwo = PSController_TX_RX (0x00); //Read digital byte 2
        PSController_TX_RX (0x00);              //Read right X analog (Dont care)
        MotorBRead = PSController_TX_RX (0x00); //Read right Y analog
        PSController_TX_RX (0x00);              //Read left X analog (Dont care)
        MotorARead = PSController_TX_RX (0x00); //Read left Y analog
        LATBbits.LATB4 = 1;                     //End transmission
```

```c
    if (MotorCEnable >= 100)            //Check for enable
    {
        LATDbits.LATD6 = 0;             //Motor C reverse
        LATDbits.LATD7 = 1;
    }
    else
    {
        LATDbits.LATD6 = 0;             //Motor C off
        LATDbits.LATD7 = 0;
    }
}
else
{
    LATDbits.LATD6 = 0;                 //Motor C off
    LATDbits.LATD7 = 0;
    if (MotorCPrev != 0)
    {
        MotorCEnable = 0;               //Reset delay
    }
    MotorCPrev = 0;
}

//Process Motor D (Suction)
if ((DigitalTwo & 0b00001000) == 0x00)
//Looks for Triangle
{
    LATDbits.LATD5 = 0;                 //Motor D forward
    LATDbits.LATD4 = 1;
    LATDbits.LATD0 = 1;                 //Pump indicator LED on
}
else if ((DigitalTwo & 0b00000001) == 0x00)
//Looks for Looks for Square
{
    LATDbits.LATD4 = 0;                 //Motor D off
    LATDbits.LATD5 = 0;
    LATDbits.LATD0 = 0;                 //Pump indicator LED off
}

//Process Motor E (Gripper)
if (((DigitalTwo & 0b00000010) == 0x00) ||
    ((DigitalTwo & 0b00100000) == 0x00))
//Looks for X or L1
{
    //Look for direction change
    if (MotorEPrev == 1)
    {
        MotorEEnable = 0;               //Reset delay
    }
    MotorEPrev = 2;                     //Update direction
```

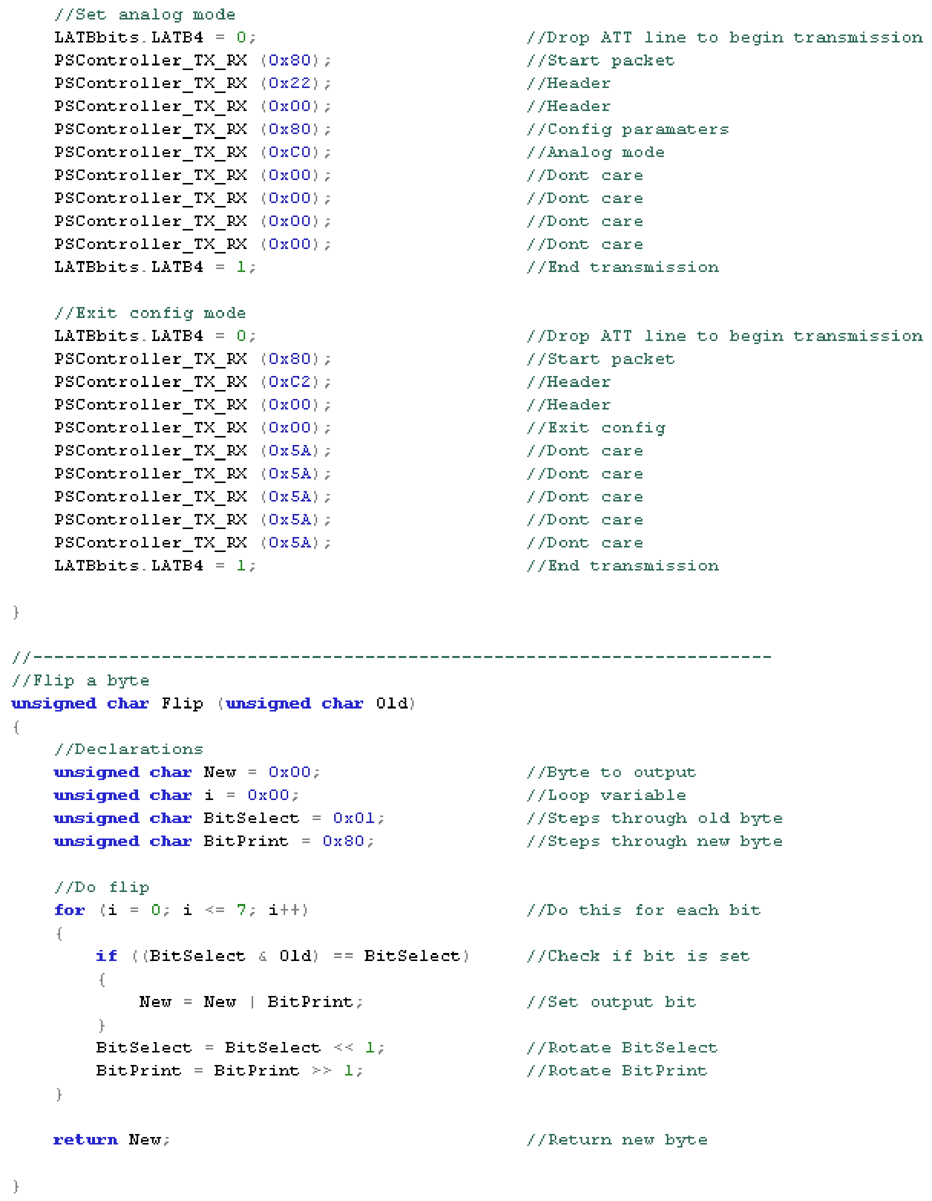
```c
//Output Motor B (Right)
//Look for sign change
if (((MotorBSpeed <= 0) && (MotorBSpeedPrev > 0)) ||
    ((MotorBSpeed >= 0) && (MotorBSpeedPrev < 0)))
{
    MotorBEnable = 0;                   //Delay
}
MotorBSpeedPrev = MotorBSpeed;          //Update Previous Speed
if (MotorBEnable >= 100)
{
    MotorBOut = MotorBSpeed;            //Use desired speed
}
else
{
    MotorBOut = 0;                      //Turn motor off
}

//Process Motor C   (Vertical)
if (((DigitalOne & 0b00001000) == 0x00) ||
    ((DigitalTwo & 0b10000000) == 0x00))
//Looks for Up or L2
{
    //Look for direction change
    if (MotorCPrev == 1)
    {
        MotorCEnable = 0;               //Reset delay
    }
    MotorCPrev = 2;                     //Update direction
    if (MotorCEnable >= 100)            //Check for enable
    {
        LATDbits.LATD7 = 0;             //Motor C forward
        LATDbits.LATD6 = 1;
    }
    else
    {
        LATDbits.LATD6 = 0;             //Motor C off
        LATDbits.LATD7 = 0;
    }
}
else if (((DigitalOne & 0b00000010) == 0x00) ||
    ((DigitalTwo & 0b01000000) == 0x00))
//Looks for Down or R2
{
    //Look for direction change
    if (MotorCPrev == 2)
    {
        MotorCEnable = 0;               //Reset delay
    }
    MotorCPrev = 1;                     //Update direction
```

```
        if (MotorBEnable >= 100)                    //Cap
        {
            MotorBEnable = 100;
        }
        MotorCEnable++;                             //Increment
        if (MotorCEnable >= 100)                    //Cap
        {
            MotorCEnable = 100;
        }
        MotorEEnable++;                             //Increment
        if (MotorEEnable >= 100)                    //Cap
        {
            MotorEEnable = 100;
        }

    }

    while (1)                                       //Dont fall out
    {}

}

//----------------------------------------------------------------------
//Setup Micorcontroller
void Setup (void)
{
    int i = 0x00;                                   //Local looping variable

    //Setup Oscilator
    OSCCON = 0b01100000;                            //Use internal OSC at 4MHZ

    //Initial behavior
    LATBbits.LATB5 = 0;                             //LEDs initially off
    LATCbits.LATC2 = 0;
    LATDbits.LATD0 = 0;
    LATDbits.LATD1 = 0;

    LATBbits.LATB4 = 1;                             //ATT line initially inactive

    LATBbits.LATB2 = 0;                             //Motor A initially off
    LATBbits.LATB3 = 0;
    LATBbits.LATB0 = 0;                             //Motor B initially off
    LATBbits.LATB1 = 0;
    LATDbits.LATD6 = 0;                             //Motor C initially off
    LATDbits.LATD7 = 0;
    LATDbits.LATD4 = 0;                             //Motor D initially off
    LATDbits.LATD5 = 0;
    LATCbits.LATC0 = 0;                             //Motor E initially off
    LATCbits.LATC1 = 0;
```

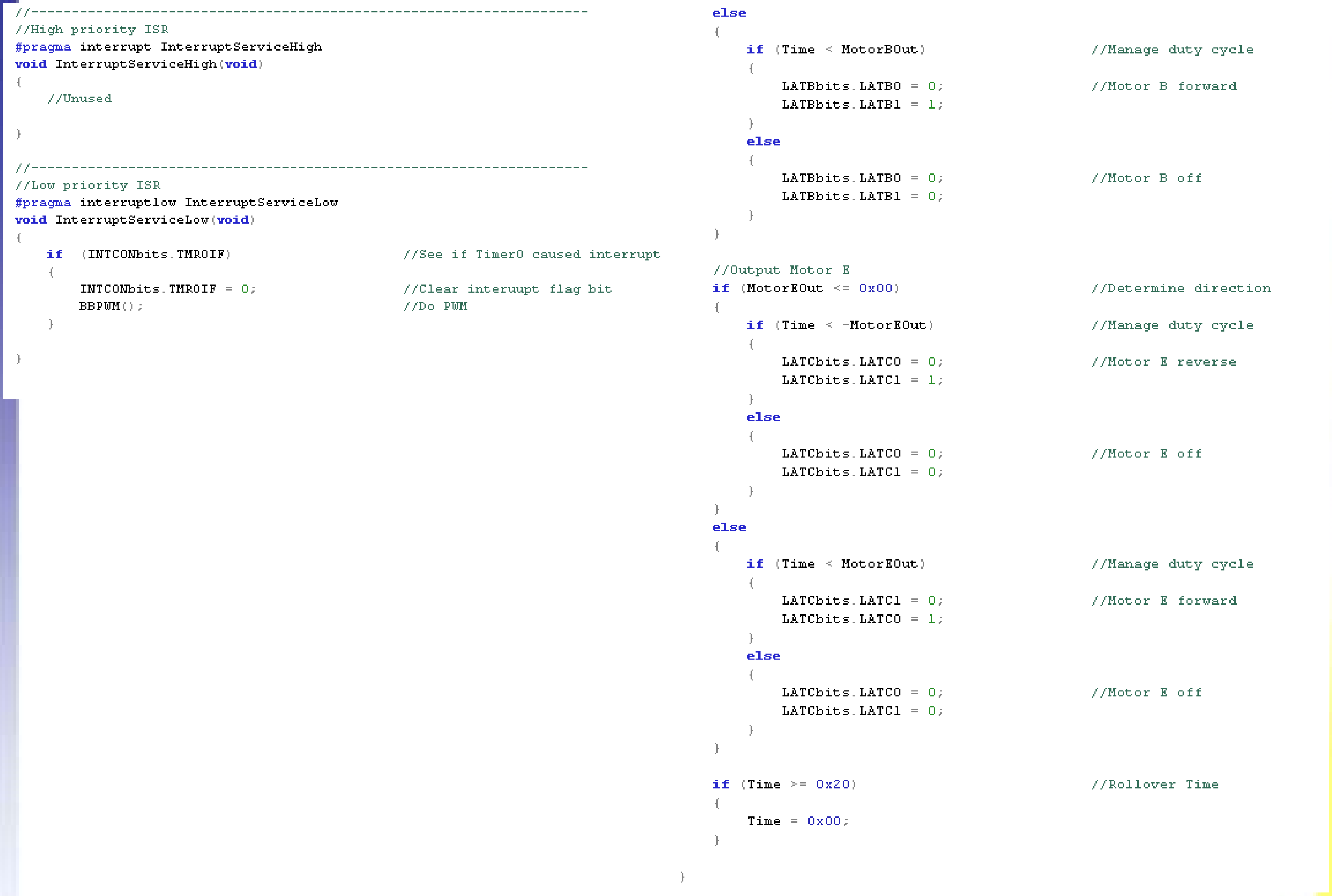
```
    if (MotorEEnable >= 100)            //Check for enable
    {
        MotorEOut = 29;                 //Gipper close
    }
    else
    {
        MotorEOut = 0;                  //Girpper Off
    }

}
else if (((DigitalTwo & 0b00000100) == 0x00) ||
    ((DigitalTwo & 0b00010000) == 0x00))
//Looks for Circle or R1
{
    //Look for direction change
    if (MotorEPrev == 2)
    {
        MotorEEnable = 0;               //Reset delay
    }
    MotorEPrev = 1;                     //Update direction
    if (MotorEEnable >= 100)            //Check for enable
    {
        MotorEOut = -29;                //Gipper open
    }
    else
    {
        MotorEOut = 0;                  //Girpper Off
    }
}
else
{
    MotorEOut = 0;                      //Gripper off
    if (MotorEPrev != 0)
    {
        MotorEEnable = 0;               //Reset
    }
    MotorEPrev = 0;
}

//Incremate enable counters
MotorAEnable++;                         //Increment
if (MotorAEnable >= 100)                //Cap
{
    MotorAEnable = 100;
}
MotorBEnable++;                         //Increment
```

```c
    T0CON = 0b01000000;                         //Prescale 1:2
    TMR0H = 0;                                  //Clear timer
    TMR0L = 0;                                  //Use upper byte
    T0CONbits.TMR0ON = 1;                       //Start timer

    //Set up global interrupts
    RCONbits.IPEN = 1;                          //Enable priority levels on interrupts
    INTCONbits.GIEL = 1;                        //Low priority interrupts allowed
    INTCONbits.GIEH = 1;                        //Interrupting enabled

}

//----------------------------------------------------------------------
//Write and Read data from PSController via SPI port
unsigned char PSController_TX_RX (unsigned char DataWrite)
{
    //Declarations
    unsigned char DataRead = 0x00;

    SSPBUF = DataWrite;                         //Send desired data to controller
    while (SSPSTATbits.BF == 0);                //Wait for end of recieve
    DataRead = SSPBUF;                          //Read new data from controller

    return DataRead;                            //Function returns data from controller

}

//----------------------------------------------------------------------
//PS controller config sequence
void PSConfig (void)
{
    //Poll packet
    LATBbits.LATB4 = 0;                         //Drop ATT line to begin transmission
    PSController_TX_RX (0x80);                  //Start packet
    PSController_TX_RX (0x42);                  //Header
    PSController_TX_RX (0x00);                  //Header
    PSController_TX_RX (0xFF);                  //Dont care
    PSController_TX_RX (0xFF);                  //Dont care
    LATBbits.LATB4 = 1;                         //End transmission

    //Enter config mode
    LATBbits.LATB4 = 0;                         //Drop ATT line to begin transmission
    PSController_TX_RX (0x80);                  //Start packet
    PSController_TX_RX (0xC2);                  //Header
    PSController_TX_RX (0x00);                  //Header
    PSController_TX_RX (0x80);                  //Enter config
    PSController_TX_RX (0x00);                  //Dont care
    LATBbits.LATB4 = 1;                         //End transmission
```

```c
//Setup Ports
TRISA = 0x00;                               //Set Port A I/O
                                            /*Port A Bit Functions:
                                              None
                                            */

TRISB = 0x00;                               //Set Port B I/O
                                            /*Port B Bit Functions:
                                              RB0: Motor B Neg
                                              RB1: Motor B Pos
                                              RB2: Motor A Neg
                                              RB3: Motor A Pos
                                              RB4: PS Controller ATT Line
                                              RB5: LED 1
                                            */

TRISC = 0x10;                               //Set Port C I/O
                                            /*Port C Bit Functions:
                                              RC0: Motor E Pos
                                              RC1: Motor E Neg
                                              RC2: LED 4
                                              RC3: SPI clock out
                                              RC4: SDI in
                                              RC5: SDO out
                                            */

TRISD = 0x00;                               //Set Port D I/O
                                            /*Port D Bit Functions:
                                              RD0: LED 3
                                              RD1: LED 4
                                              RD4: Motor D Neg
                                              RD5: Motor D Pos
                                              RD6: Motor C Neg
                                              RD7: Motor C Pos
                                            */

//Setup SPI port as master
SSPSTAT = 0b00000000;                       //Setup status register
SSPCON1 = 0b00110010;                       //Setup control register

//Configure controller for analog mode
for (i = 0; i <= 3; i++)                    //3x for redundancy
{
    PSConfig();
}

//Set up Interrupt for timer
INTCONbits.TMR0IF = 0;                      //Clear roll-over interrupt flag
INTCON2bits.TMR0IP = 0;                     //Timer0 is low priority interrupt
INTCONbits.TMR0IE = 1;                      //Enable the Timer0 interrupt.
```



```
    //Set analog mode
    LATBbits.LATB4 = 0;                        //Drop ATT line to begin transmission
    PSController_TX_RX (0x80);                 //Start packet
    PSController_TX_RX (0x22);                 //Header
    PSController_TX_RX (0x00);                 //Header
    PSController_TX_RX (0x80);                 //Config paramaters
    PSController_TX_RX (0xC0);                 //Analog mode
    PSController_TX_RX (0x00);                 //Dont care
    PSController_TX_RX (0x00);                 //Dont care
    PSController_TX_RX (0x00);                 //Dont care
    PSController_TX_RX (0x00);                 //Dont care
    LATBbits.LATB4 = 1;                        //End transmission

    //Exit config mode
    LATBbits.LATB4 = 0;                        //Drop ATT line to begin transmission
    PSController_TX_RX (0x80);                 //Start packet
    PSController_TX_RX (0xC2);                 //Header
    PSController_TX_RX (0x00);                 //Header
    PSController_TX_RX (0x00);                 //Exit config
    PSController_TX_RX (0x5A);                 //Dont care
    PSController_TX_RX (0x5A);                 //Dont care
    PSController_TX_RX (0x5A);                 //Dont care
    PSController_TX_RX (0x5A);                 //Dont care
    PSController_TX_RX (0x5A);                 //Dont care
    LATBbits.LATB4 = 1;                        //End transmission

}

//----------------------------------------------------------------------
//Flip a byte
unsigned char Flip (unsigned char Old)
{
    //Declarations
    unsigned char New = 0x00;                  //Byte to output
    unsigned char i = 0x00;                    //Loop variable
    unsigned char BitSelect = 0x01;            //Steps through old byte
    unsigned char BitPrint = 0x80;             //Steps through new byte

    //Do flip
    for (i = 0; i <= 7; i++)                   //Do this for each bit
    {
        if ((BitSelect & Old) == BitSelect)    //Check if bit is set
        {
            New = New | BitPrint;              //Set output bit
        }
        BitSelect = BitSelect << 1;            //Rotate BitSelect
        BitPrint = BitPrint >> 1;              //Rotate BitPrint
    }

    return New;                                //Return new byte

}
```



```
//---------------------------------------------------------------------
//High priority ISR
#pragma interrupt InterruptServiceHigh
void InterruptServiceHigh(void)
{
    //Unused

}

//---------------------------------------------------------------------
//Low priority ISR
#pragma interruptlow InterruptServiceLow
void InterruptServiceLow(void)
{
    if  (INTCONbits.TMR0IF)                    //See if Timer0 caused interrupt
    {
        INTCONbits.TMR0IF = 0;                 //Clear interuupt flag bit
        BBPWM();                               //Do PWM
    }

}
```

```
    else
    {
        if (Time < MotorBOut)                           //Manage duty cycle
        {
            LATBbits.LATB0 = 0;                         //Motor B forward
            LATBbits.LATB1 = 1;
        }
        else
        {
            LATBbits.LATB0 = 0;                         //Motor B off
            LATBbits.LATB1 = 0;
        }
    }

    //Output Motor E
    if (MotorEOut <= 0x00)                              //Determine direction
    {
        if (Time < -MotorEOut)                          //Manage duty cycle
        {
            LATCbits.LATC0 = 0;                         //Motor E reverse
            LATCbits.LATC1 = 1;
        }
        else
        {
            LATCbits.LATC0 = 0;                         //Motor E off
            LATCbits.LATC1 = 0;
        }
    }
    else
    {
        if (Time < MotorEOut)                           //Manage duty cycle
        {
            LATCbits.LATC1 = 0;                         //Motor E forward
            LATCbits.LATC0 = 1;
        }
        else
        {
            LATCbits.LATC0 = 0;                         //Motor E off
            LATCbits.LATC1 = 0;
        }
    }

    if (Time >= 0x20)                                   //Rollover Time
    {
        Time = 0x00;
    }

}
```